



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ
ΥΠΟΛΟΓΙΣΤΩΝ

ΤΡΙΤΗ ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΝΑΦΟΡΑ
ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

Ομάδα 25
Αναστασία Χριστίνα Λίβα
03119029
Γεώργιος Μυστριώτης
03119065

Περιεχόμενα

Άσκηση 1	2
Άσκηση 2	8
Άσκηση 3	15

Άσκηση 1

Ακολουθεί ο πηγαίος κώδικας για την πρώτη άσκηση.

```
/*
 * simplesync.c
 *
 * A simple synchronization exercise.
 *
 * Vangelis Koukis <vkoukis@cslab.ece.ntua.gr>
 * Operating Systems course, ECE, NTUA
 */

#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <time.h>

/*
 * POSIX thread functions do not return error numbers in errno,
 * but in the actual return value of the function call instead.
 * This macro helps with error reporting in this case.
 */
#define perror_pthread(ret, msg) \
    do { errno = ret; perror(msg); } while (0)

#define N 10000000

/* Dots indicate lines where you are free to insert code at will */
/* ... */
#if defined(SYNC_ATOMIC) ^ defined(SYNC_MUTEX) == 0
# error You must #define exactly one of SYNC_ATOMIC or SYNC_MUTEX.
#endif

#if defined(SYNC_ATOMIC)
# define USE_ATOMIC_OPS 1
#else
# define USE_ATOMIC_OPS 0
```

```
#endif

pthread_mutex_t lock;

void *increase_fn(void *arg)
{
    int i;
    volatile int *ip = arg;

    fprintf(stderr, "About to increase
variable %d times\n", N);
    for (i = 0; i < N; i++) {
        if (USE_ATOMIC_OPS) {
            /*...*/
            /* You can modify the
following line */
            __sync_add_and_fetch(ip, 1);
            /* ... */
        } else {
            pthread_mutex_lock(&lock);
            /* You cannot modify
the following line */
            ++(*ip);
            pthread_mutex_unlock(&lock);
        }
    }
    fprintf(stderr, "Done increasing variable.\n");

    return NULL;
}

void *decrease_fn(void *arg)
{
    int i;
    volatile int *ip = arg;

    fprintf(stderr, "About to decrease
variable %d times\n", N);
    for (i = 0; i < N; i++) {
        if (USE_ATOMIC_OPS) {
```

```
        /* ... */
        /* You can modify
        the following line */
        __sync_sub_and_fetch(ip, 1);
        /* ... */
    } else {
        pthread_mutex_lock(&lock);
        /* You cannot modify
        the following line */
        --(*ip);
        pthread_mutex_unlock(&lock);
    }
}
fprintf(stderr, "Done decreasing variable.\n");

return NULL;
}

int main(int argc, char *argv[])
{
    time_t begin = time(NULL);
    int val, ret, ok;
    pthread_t t1, t2;

    /*
     * Initial value
     */
    val = 0;

    /*
     * Create threads
     */
    ret = pthread_create(&t1, NULL, increase_fn, &val);
    if (ret) {
        perror_pthread(ret, "pthread_create");
        exit(1);
    }
    ret = pthread_create(&t2, NULL, decrease_fn, &val);
    if (ret) {
```

```
        perror_thread(ret, "pthread_create");
        exit(1);
    }

    /*
     * Wait for threads to terminate
     */
    ret = pthread_join(t1, NULL);
    if (ret)
        perror_thread(ret, "pthread_join");
    ret = pthread_join(t2, NULL);
    if (ret)
        perror_thread(ret, "pthread_join");

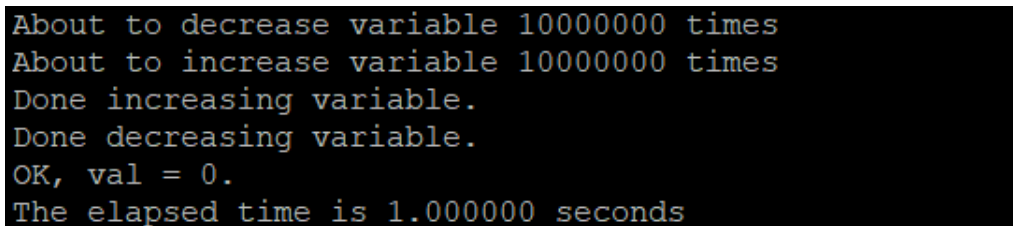
    /*
     * Is everything OK?
     */
    ok = (val == 0);

    printf("%sOK, val = %d.\n", ok ? "" : "NOT ", val);

    time_t end = time(NULL);

    printf("The elapsed time is %f\n", (float)(end - begin));

    return ok;
}
```

A terminal window with a black background and green text. The output shows the program's execution: it prints 'About to decrease variable 10000000 times', 'About to increase variable 10000000 times', 'Done increasing variable.', 'Done decreasing variable.', 'OK, val = 0.', and 'The elapsed time is 1.000000 seconds'.

```
About to decrease variable 10000000 times
About to increase variable 10000000 times
Done increasing variable.
Done decreasing variable.
OK, val = 0.
The elapsed time is 1.000000 seconds
```

Αποτέλεσμα κώδικα simplesync-atomic

```
About to increase variable 10000000 times
About to decrease variable 10000000 times
Done decreasing variable.
Done increasing variable.
OK, val = 0.
The elapsed time is 27.000000 seconds
```

Αποτέλεσμα κώδικα simplesync-mutex

Ερώτηση 1 Χωρίς συγχρονισμό το πρόγραμμα τελειώνει γρηγορότερα, με αμφίβολα ωστόσο αποτελέσματα. Αυτό συμβαίνει γιατί χωρίς συγχρονισμό κάθε thread δεν περιμένει το άλλο, αλλά τρέχουν ταυτόχρονα. Κατά συνέπεια σε περίπτωση επεξεργασίας κοινών δεδομένων υπάρχει περίπτωση να μη διαβαστεί σωστή τιμή από κάποιο νήμα.

Ερώτηση 2 Με χρήση ατομικών λειτουργιών του gcc είναι γρηγορότερο από ότι με mutexes καθώς όπως θα δούμε και στα επόμενα ερωτήματα για την εκτέλεση ατομικών λειτουργιών χρειάζεται μόνο μία εντολή ενώ για τα mutexes ολόκληρη συνάρτηση.

Ερώτηση 3

```
.loc 1 52 0
movq    -16(%rbp), %rax
lock addl    $1, (%rax)
.loc 1 48 0
addl    $1, -4(%rbp)
```

Όπως προειπώθηκε έχουμε μόνο μία εντολή assembly

Ερώτηση 4

```
.loc 1 55 0
movl    $lock, %edi
call    pthread_mutex_lock
.loc 1 57 0
movq    -16(%rbp), %rax
movl    (%rax), %eax
leal    1(%rax), %edx
movq    -16(%rbp), %rax
movl    %edx, (%rax)
.loc 1 58 0
movl    $lock, %edi
call    pthread_mutex_unlock
.loc 1 48 0
addl    $1, -4(%rbp)
```

Όπως προειπώθηκε έχουμε συνάρτηση assembly

```
assembly-atomic.s: simplesync.c
$(CC) $(CFLAGS) -S -g -DSYNC_ATOMIC -c -o assembly-atomic.s simplesync.c

assembly-mutex.s: simplesync.c
$(CC) $(CFLAGS) -S -g -DSYNC_MUTEX -c -o assembly-mutex.s simplesync.c
```

Για τα δύο προηγούμενα ερωτήματα εκτελέστηκαν οι δύο παραπάνω εντολές αντίστοιχα, οι οποίες προστέθηκαν στο Makefile.

Άσκηση 2

Ακολουθεί ο πηγαίος κώδικας για την δεύτερη άσκηση.

```
/*
 * mandel.c
 *
 * A program to draw the Mandelbrot Set on a 256-color xterm.
 *
 */

#include <stdio.h>
#include <unistd.h>
#include <assert.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>

#include <pthread.h>
#include "mandel-lib.h"
#include <semaphore.h>
#include <errno.h>

#define perror_pthread(ret, msg) \
    do { errno = ret; perror(msg); } while (0)

#define MANDEL_MAX_ITERATION 100000

/*****
 * Compile-time parameters *
 *****/

/*
 * Output at the terminal is is x_chars wide by y_chars long
 */
sem_t *sem;
int y_chars = 50;
int x_chars = 90;

/*
```

```
* The part of the complex plane to be drawn:
* upper left corner is (xmin, ymax),
* lower right corner is (xmax, ymin)
*/
double xmin = -1.8, xmax = 1.0;
double ymin = -1.0, ymax = 1.0;

/*
 * Every character in the final output is
 * xstep x ystep units wide on the complex plane.
 */
double xstep;
double ystep;

struct pair{
    int thrcount;
    int N;
};

/*
 * This function computes a line of output
 * as an array of x_char color values.
 */
void compute_mandel_line(int line, int color_val[])
{
    /*
     * x and y traverse the complex plane.
     */
    double x, y;

    int n;
    int val;

    /* Find out the y value corresponding to this line */
    y = ymax - ystep * line;

    /* and iterate for all points on this line */
    for (x = xmin, n = 0; n < x_chars; x+= xstep, n++) {

        /* Compute the point's color value */
```

```
        val = mandel_iterations_at_point(x, y,
        MANDEL_MAX_ITERATION);
        if (val > 255)
            val = 255;

        /* And store it in the color_val[] array */
        val = xterm_color(val);
        color_val[n] = val;
    }
}

/*
 * This function outputs an array of x_char color values
 * to a 256-color xterm.
 */

void output_mandel_line(int fd, int color_val[])
{
    int i;

    char point = '@';
    char newline = '\n';

    for (i = 0; i < x_chars; i++) {
        /* Set the current color, then output the point */
        set_xterm_color(fd, color_val[i]);
        if (write(fd, &point, 1) != 1) {
            perror("compute_and_output_mandel_line:
            write point");
            exit(1);
        }
    }

    /* Now that the line is done, output a newline character */
    if (write(fd, &newline, 1) != 1) {
        perror("compute_and_output_mandel_line:
        write newline");
        exit(1);
    }
}
```

```
    }  
}  
  
void* compute_and_output_mandel_line(void *arg)  
{  
    int fd=1;  
    volatile struct pair *pair = arg;  
    int thrcount = pair->thrcount;  
    int N = pair->N;  
    int i;  
    for(i=thrcount;i<y_chars;i+=N){  
        /*  
         * A temporary array, used to hold  
         * color values for the line being drawn  
         */  
        // sem_wait(sem+step);  
        int color_val[x_chars];  
        // sem_wait(sem+step);  
        // printf("line = %d\n", line);  
        compute_mandel_line(i, color_val);  
        sem_wait(sem+thrcount);  
        output_mandel_line(fd, color_val);  
        if(thrcount==N-1){  
            sem_post(sem);  
        }  
        else{  
            sem_post(sem+thrcount+1);  
        }  
    }  
    return NULL;  
}  
  
int main(int argc, char *argv[])  
{  
    int i,N;  
  
    int ret;  
  
    N=atoi(argv[1]);
```

```
struct pair pair[N];

pthread_t thr[N];

sem_t semarr[N];
sem=semarr;

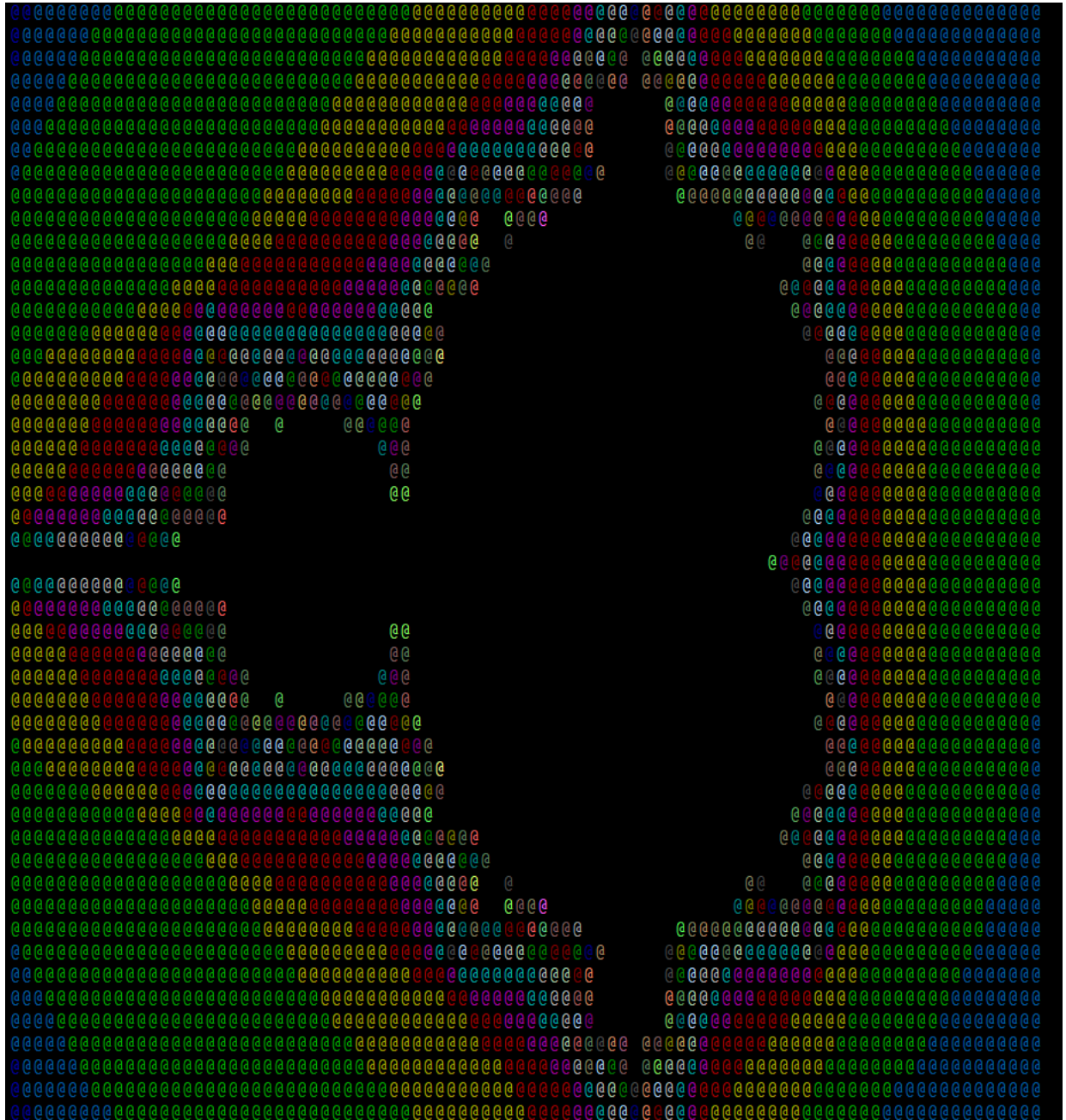
for (i=1;i<N;i++){
    sem_init(sem+i, 0 , 0);
}
sem_init (sem, 0, 1);

xstep = (xmax - xmin) / x_chars;
ystep = (ymax - ymin) / y_chars;

/*
 * draw the Mandelbrot Set, one line at a time.
 * Output is sent to file descriptor
 * '1', i.e., standard output.
 */
for(i=0;i<N;i++){
    pair[i].thrcount=i;
    pair[i].N=N;
    ret=pthread_create(&thr[i], NULL,
        compute_and_output_mandel_line,&pair[i]);
    if (ret) {
        perror_pthread(ret, "pthread_create");
        exit(1);
    }
}

for (i=0; i<N; i++){
    pthread_join (thr[i], NULL);
}

reset_xterm_color(1);
return 0;
}
```



Αποτέλεσμα κώδικα για 4 νήματα

Ερώτηση 1 Χρειάζονται N σημαφόροι, όσα και τα νήματα που τρέχουν παράλληλα ώστε ο σημαφόρος του καθε νήματος όταν αυτό ολοκληρώνει το τύπωμα της γραμμής που του αντιστοιχεί να ενεργοποιεί τον επόμενο σημαφόρο για ώστε το αντίστοιχο νήμα να τυπώσει την επόμενη γράμμη.

Ερώτηση 2 Για ένα νήμα (σειριακό πρόγραμμα) απαιτείται λίγο περισσότερο από ένα δευτερόλεπτο ώστε να ολοκληρωθεί η εκτέλεση του προγράμματος. Με χρήση δύο νημάτων (παράλληλο πρόγραμμα) η εκτέλεση ολοκληρώνεται σε περίπου μισό δευτερόλεπτο.

Ερώτηση 3 Το παράλληλο πρόγραμμα που φτιάξαμε εμφανίζει επιτάχυνση ενώ το κρίσιμο τμήμα πρέπει να περιέχει μόνο τη φάση εξόδου κάθε γραμμής, αφού ο υπολογισμός των γραμμών γίνεται ανεξάρτητα.

Ερώτηση 4 Για να επανέλθει το χρώμα των γραμμών εκτελούμε την εντολή `reset_xterm_color(1)` κάθε φορά μετά την εκτύπωση μιας γραμμής.

Άσκηση 3

Ακολουθεί ο πηγαίος κώδικας για την τρίτη άσκηση.

```
/*
 * kgarten.c
 *
 * A kindergarten simulator.
 * Bad things happen if teachers and children
 * are not synchronized properly.
 *
 *
 * Author:
 * Vangelis Koukis <vkoukis@cslab.ece.ntua.gr>
 *
 * Additional Authors:
 * Stefanos Gerangelos <sgerag@cslab.ece.ntua.gr>
 * Anastassios Nanos <ananos@cslab.ece.ntua.gr>
 * Operating Systems course, ECE, NTUA
 *
 */

#include <time.h>
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <semaphore.h>

/*
 * POSIX thread functions do not return error numbers in errno,
 * but in the actual return value of the function call instead.
 * This macro helps with error reporting in this case.
 */
#define perror_pthread(ret, msg) \
    do { errno = ret; perror(msg); } while (0)

/* A virtual kindergarten */
struct kgarten_struct {
```



```
/*
 * Here you may define any mutexes /
 * condition variables / other variables
 * you may need.
 */

pthread_mutex_t lock;
pthread_cond_t co;

/*
 * You may NOT modify anything in the
 * structure below this
 * point.
 */
int vt;
int vc;
int ratio;

pthread_mutex_t mutex;
};

/*
 * A (distinct) instance of this structure
 * is passed to each thread
 */
struct thread_info_struct {
    pthread_t tid; /* POSIX thread
    id, as returned by the library */

    struct kgarten_struct *kg;
    int is_child; /* Nonzero if
    this thread simulates children, zero otherwise */

    int thrid; /* Application-defined thread id */
    int thrcnt;
    unsigned int rseed;
};

int safe_atoi(char *s, int *val)
{
```

```
    long l;
    char *endp;

    l = strtol(s, &endp, 10);
    if (s != endp && *endp == '\\0') {
        *val = l;
        return 0;
    } else
        return -1;
}

void *safe_malloc(size_t size)
{
    void *p;

    if ((p = malloc(size)) == NULL) {
        fprintf(stderr, "Out of memory,
        failed to allocate %zd bytes\\n",
            size);
        exit(1);
    }

    return p;
}

void usage(char *argv0)
{
    fprintf(stderr, "Usage: %s thread_count
    child_threads c_t_ratio\\n\\n"
        "Exactly two argument required:\\n"
        "    thread_count: Total number
        of threads to create.\\n"
        "    child_threads: The number
        of threads simulating children.\\n"
        "    c_t_ratio: The allowed ratio of
        children to teachers.\\n\\n",
        argv0);
    exit(1);
}
```

```
void bad_thing(int thrid, int children, int teachers)
{
    int thing, sex;
    int namecnt, nameidx;
    char *name, *p;
    char buf[1024];

    char *things[] = {
        "Little %s put %s finger
        in the wall outlet and got electrocuted!",
        "Little %s fell off the slide
        and broke %s head!",
        "Little %s was playing with
        matches and lit %s hair on fire!",
        "Little %s drank a bottle of
        acid with %s lunch!",
        "Little %s caught %s hand
        in the paper shredder!",
        "Little %s wrestled with
        a stray dog and it bit %s finger off!"
    };

    char *boys[] = {
        "George", "John", "Nick", "Jim",
        "Constantine", "Chris", "Peter", "Paul",
        "Steve", "Billy", "Mike",
        "Vangelis", "Antony"
    };

    char *girls[] = {
        "Maria", "Irene", "Christina", "Helena",
        "Georgia", "Olga", "Sophie", "Joanna",
        "Zoe", "Catherine", "Marina", "Stella",
        "Vicky", "Jenny"
    };

    thing = rand() % 4;
    sex = rand() % 2;

    namecnt = sex ? sizeof(boys)/sizeof(boys[0]) :
    sizeof(girls)/sizeof(girls[0]);
    nameidx = rand() % namecnt;
```

```
    name = sex ? boys[nameidx] : girls[nameidx];

    p = buf;
    p += sprintf(p, "*** Thread %d: Oh no! ", thrid);
    p += sprintf(p, things[thing], name, sex ? "his" : "her");
    p += sprintf(p, "\n*** Why were there only
                    %d teachers for %d children?!\n",
                    teachers, children);

    /* Output everything in a single atomic call */
    printf("%s", buf);
}

void child_enter(struct thread_info_struct *thr)
{
    pthread_mutex_lock(&thr->kg->lock);
    while (thr->kg->vt * thr->kg->ratio < thr->kg->vc + 1) {
        pthread_cond_wait ( &thr->kg->co , &thr->kg->lock);
    }
    if (!thr->is_child) {
        fprintf(stderr, "Internal error: %s
                        called for a Teacher thread.\n",
                        __func__);
        exit(1);
    }

    fprintf(stderr, "THREAD %d: CHILD ENTER\n", thr->thrid);

    //    pthread_mutex_lock(&thr->kg->mutex);
    //    ++(thr->kg->vc);
    //    pthread_mutex_unlock(&thr->kg->mutex);
    //    pthread_mutex_unlock(&thr->kg->lock);
}

void child_exit(struct thread_info_struct *thr)
{
    pthread_mutex_lock(&thr->kg->lock);

    if (!thr->is_child) {
        fprintf(stderr, "Internal error:
                        %s called for a Teacher thread.\n",
                        __func__);
    }
}
```

```
        exit(1);
    }

    fprintf(stderr, "THREAD %d: CHILD EXIT\n", thr->thrid);

//    pthread_mutex_lock(&thr->kg->mutex);
//    --(thr->kg->vc);
//    pthread_cond_broadcast (&thr->kg->co);
//    pthread_mutex_unlock(&thr->kg->mutex);
//    pthread_mutex_unlock(&thr->kg->lock);
//}

void teacher_enter(struct thread_info_struct *thr)
{
    pthread_mutex_lock(&thr->kg->lock);
    if (thr->is_child) {
        fprintf(stderr, "Internal error: %s
        called for a Child thread.\n",
            __func__);
        exit(1);
    }

    fprintf(stderr, "THREAD %d: TEACHER ENTER\n", thr->thrid);

//    pthread_mutex_lock(&thr->kg->mutex);
//    ++(thr->kg->vt);
//    pthread_cond_broadcast(&thr->kg->co);
//    pthread_mutex_unlock(&thr->kg->mutex);
//    pthread_mutex_unlock(&thr->kg->lock);
//}

void teacher_exit(struct thread_info_struct *thr)
{
    pthread_mutex_lock(&thr->kg->lock);
    while ( (thr->kg->vt -1) * thr->kg->ratio < thr->kg->vc) {
        pthread_cond_wait(&thr->kg->co, &thr->kg->lock);
    }
    if (thr->is_child) {
        fprintf(stderr, "Internal error: %s
        called for a Child thread.\n",
            __func__);
    }
}
```

```
        exit(1);
    }

    fprintf(stderr, "THREAD %d: TEACHER EXIT\n", thr->thrid);

//    pthread_mutex_lock(&thr->kg->mutex);
//    --(thr->kg->vt);
//    pthread_mutex_unlock(&thr->kg->mutex);
//    pthread_mutex_unlock(&thr->kg->lock);
}

/*
 * Verify the state of the kindergarten.
 */
void verify(struct thread_info_struct *thr)
{
    struct kgarten_struct *kg = thr->kg;
    int t, c, r;

    c = kg->vc;
    t = kg->vt;
    r = kg->ratio;

    fprintf(stderr, "                Thread %d: Teachers:
%d, Children: %d\n",
            thr->thrid, t, c);

    if (c > t * r) {
        bad_thing(thr->thrid, c, t);
        exit(1);
    }
}

/*
 * A single thread.
 * It simulates either a teacher, or a child.
 */
void *thread_start_fn(void *arg)
{
```

```
/* We know arg points to an instance of thread_info_struct */
struct thread_info_struct *thr = arg;
char *nstr;

fprintf(stderr, "Thread %d of %d. START.\n",
thr->thrid, thr->thrcnt);

nstr = thr->is_child ? "Child" : "Teacher";
for (;;) {
    fprintf(stderr, "Thread %d [%s]: Entering.\n",
thr->thrid, nstr);
    if (thr->is_child)
        child_enter(thr);
    else
        teacher_enter(thr);

    fprintf(stderr, "Thread %d [%s]: Entered.\n",
thr->thrid, nstr);

    /*
     * We're inside the critical section,
     * just sleep for a while.
     */
    /* usleep(rand_r(&thr->rseed) % 1000000 /
(thr->is_child ? 10000 : 1)); */
    pthread_mutex_lock(&thr->kg->mutex);
    verify(thr);
    pthread_mutex_unlock(&thr->kg->mutex);

    usleep(rand_r(&thr->rseed) % 1000000);

    fprintf(stderr, "Thread %d [%s]: Exiting.\n",
thr->thrid, nstr);
    /* CRITICAL SECTION END */

    if (thr->is_child)
        child_exit(thr);
    else
        teacher_exit(thr);
}
```

```
        fprintf(stderr, "Thread %d [%s]: Exited.\n",
        thr->thrid, nstr);

        /* Sleep for a while before re-entering */
        /* usleep(rand_r(&thr->rseed) % 100000 *
        (thr->is_child ? 100 : 1)); */
        usleep(rand_r(&thr->rseed) % 100000);

        pthread_mutex_lock(&thr->kg->mutex);
        verify(thr);
        pthread_mutex_unlock(&thr->kg->mutex);
    }

    fprintf(stderr, "Thread %d of %d. END.\n",
    thr->thrid, thr->thrcnt);

    return NULL;
}

int main(int argc, char *argv[])
{
    int i, ret, thrcnt, chldcnt, ratio;
    struct thread_info_struct *thr;
    struct kgarten_struct *kg;

    /*
     * Parse the command line
     */
    if (argc != 4)
        usage(argv[0]);
    if (safe_atoi(argv[1], &thrcnt) < 0 || thrcnt <= 0) {
        fprintf(stderr, "'%s' is not valid for
        'thread_count'\n", argv[1]);
        exit(1);
    }
    if (safe_atoi(argv[2], &chldcnt) < 0 ||
    chldcnt < 0 || chldcnt > thrcnt) {
        fprintf(stderr, "'%s' is not valid for
        'child_threads'\n", argv[2]);
```



```
        exit(1);
    }
    if (safe_atoi(argv[3], &ratio) < 0 || ratio < 1) {
        fprintf(stderr, "'%s' is not valid for
        'c_t_ratio'\n", argv[3]);
        exit(1);
    }

    /*
     * Initialize kindergarten and random number generator
     */
    srand(time(NULL));

    kg = safe_malloc(sizeof(*kg));
    kg->vt = kg->vc = 0;
    kg->ratio = ratio;

    ret = pthread_mutex_init(&kg->mutex, NULL);
    if (ret) {
        perror_thread(ret, "pthread_mutex_init");
        exit(1);
    }
    ret = pthread_mutex_init(&kg->lock, NULL);
    if (ret) {
        perror_thread(ret, "pthread_mutex_init");
        exit(1);
    }
    ret = pthread_cond_init(&kg->co, NULL);
    if (ret) {
        perror_thread(ret, "pthread_mutex_init");
        exit(1);
    }

    /* ... */

    /*
     * Create threads
     */
```

```
thr = safe_malloc(thrcnt * sizeof(*thr));

for (i = 0; i < thrcnt; i++) {
    /* Initialize per-thread structure */
    thr[i].kg = kg;
    thr[i].thrid = i;
    thr[i].thrcnt = thrcnt;
    thr[i].is_child = (i < chldcnt);
    thr[i].rseed = rand();

    /* Spawn new thread */
    ret = pthread_create(&thr[i].tid, NULL,
        thread_start_fn, &thr[i]);
    if (ret) {
        perror_pthread(ret, "pthread_create");
        exit(1);
    }
}

/*
 * Wait for all threads to terminate
 */
for (i = 0; i < thrcnt; i++) {
    ret = pthread_join(thr[i].tid, NULL);
    if (ret) {
        perror_pthread(ret, "pthread_join");
        exit(1);
    }
}

printf("OK.\n");

return 0;
}
```

```
Thread 2 [Child]: Exiting.
THREAD 2: CHILD EXIT
Thread 2 [Child]: Exited.
Thread 7 [Teacher]: Exiting.
THREAD 7: TEACHER EXIT
Thread 0 [Child]: Exiting.
THREAD 0: CHILD EXIT
Thread 0 [Child]: Exited.
    Thread 2: Teachers: 2, Children: 5
Thread 2 [Child]: Entering.
THREAD 2: CHILD ENTER
Thread 2 [Child]: Entered.
    Thread 2: Teachers: 2, Children: 6
    Thread 7: Teachers: 2, Children: 6
Thread 7 [Teacher]: Entering.
THREAD 7: TEACHER ENTER
Thread 7 [Teacher]: Entered.
    Thread 7: Teachers: 3, Children: 6
    Thread 0: Teachers: 3, Children: 6
Thread 0 [Child]: Entering.
THREAD 0: CHILD ENTER
Thread 0 [Child]: Entered.
    Thread 0: Teachers: 3, Children: 7
Thread 2 [Child]: Exiting.
THREAD 2: CHILD EXIT
Thread 2 [Child]: Exited.
    Thread 2: Teachers: 3, Children: 6
Thread 2 [Child]: Entering.
THREAD 2: CHILD ENTER
Thread 2 [Child]: Entered.
    Thread 2: Teachers: 3, Children: 7
Thread 6 [Child]: Exiting.
THREAD 6: CHILD EXIT
Thread 6 [Child]: Exited.
Thread 4 [Child]: Exiting.
THREAD 4: CHILD EXIT
Thread 4 [Child]: Exited.
    Thread 4: Teachers: 3, Children: 5
Thread 4 [Child]: Entering.
THREAD 4: CHILD ENTER
Thread 4 [Child]: Entered.
    Thread 4: Teachers: 3, Children: 6
Thread 0 [Child]: Exiting.
THREAD 0: CHILD EXIT
Thread 0 [Child]: Exited.
Thread 9 [Teacher]: Exiting.
THREAD 9: TEACHER EXIT
Thread 9 [Teacher]: Exited.
```

Απόσπασμα αποτελέσματος kgarten για 10, 7, 3

Ερώτηση 1 Στο σχήμα συγχρονισμού μας τα νέα παιδιά για να μπουν στο χώρο περιμένουν μέχρι να επαρκούν οι δάσκαλοι για αυτά και όσα άλλα περιμένουν νωρίτερα.

Ερώτηση 2 Στη δική μας υλοποίηση του προβλήματος δεν έχουμε καταστάσεις συναγωνισμού καθώς τα κρίσιμα τμήματα είναι κλειδωμένα μέσω mutexes.